Building Safe and Secure Systems with AADL

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Julien Delange
AADL Meeting February 15

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
15 FEB 2015	**N/A**	

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Building Safe and Secure Systems with AADL	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
Delange /Julien	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited.

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**SAR**	**30**	

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0002054

Agenda

Introduction to AADL

AADL modeling patterns for safety and security

AADL validation tools dedicated to security and safety

Demonstration

Agenda

Introduction to AADL

AADL modeling patterns for safety and security

AADL validation tools dedicated to security and safety

Demonstration

Introduction

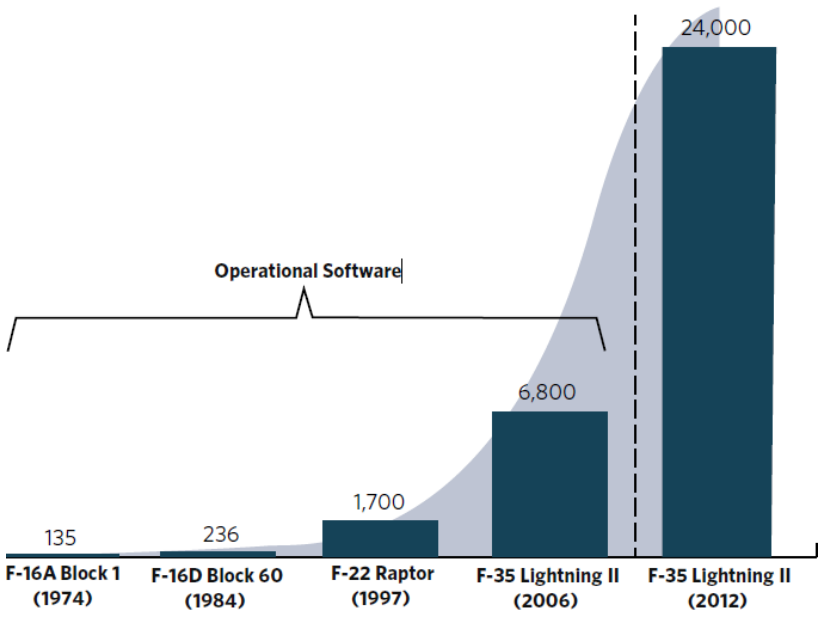
Systems are becoming extremely software-reliant

Need to verify and validate requirements

- Requirements errors propagate through design
- Need to verify/validate requirements

Major integration and coding issues

- Incur massive re-engineering rework
- Could be removed by early analysis



Architecture Analysis and Design Language

Model-Based Engineering with AADL

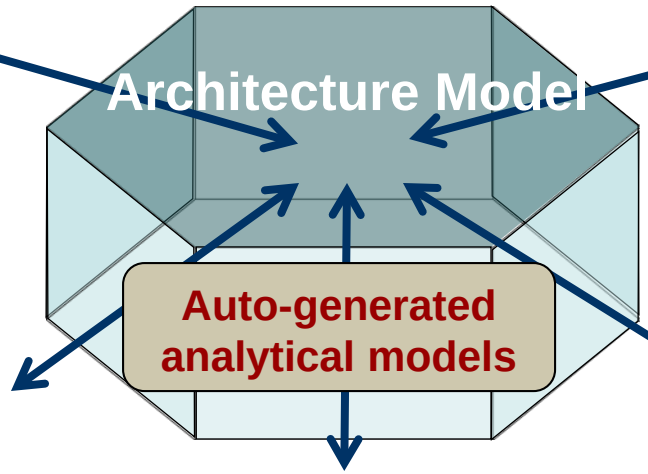
- Architecture Language Description standardized by SAE
- Description of Systems and Software Concerns
- Precise & unambiguous semantics
- Textual and Graphical Representation

Support for Model Analysis

- Verify system requirements (i.e. latency, safety)
- Check model integration before producing the implementation

AADL Model-Based Technology Overview

Single Source Annotated Architecture Model with Well-defined Semantics



Safety & Reliability
- MTBF
- FMEA
- Hazard analysis

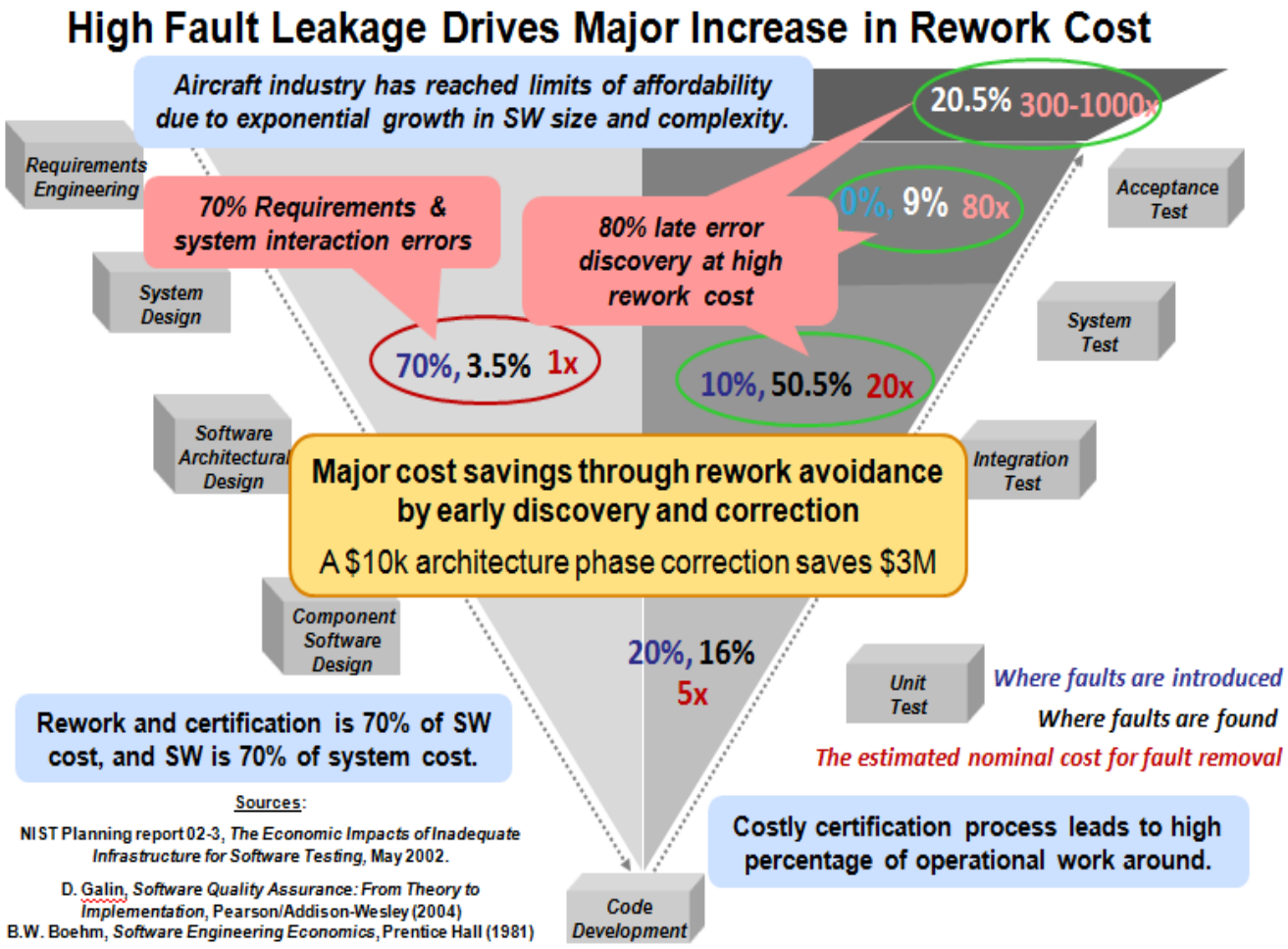
Security
- Intrusion
- Integrity
- Confidentiality

Data Quality
- Data precision/ accuracy
- Temporal correctness
- Confidence

Real-time Performance
- Execution time/ Deadline
- Deadlock/starvation
- Latency

Resource Consumption
- Bandwidth
- CPU time
- Power consumption

Understanding Actual Software Issues



Use of AADL in Development Process

Software and Component Design

- Define components requirements & interfaces
- Early verification validation of components integration

Code Development

- Auto-Generate Code (AADL, Simulink, SCADE)
- Avoid traditional coding errors
- Ensure correct translation of requirements

Unit & Integration Test

- Automatic generation of tests from models
- Reduce tests as system was validated earlier

Agenda

Introduction to AADL

AADL modeling patterns for safety and security

AADL validation tools dedicated to security and safety

Demonstration

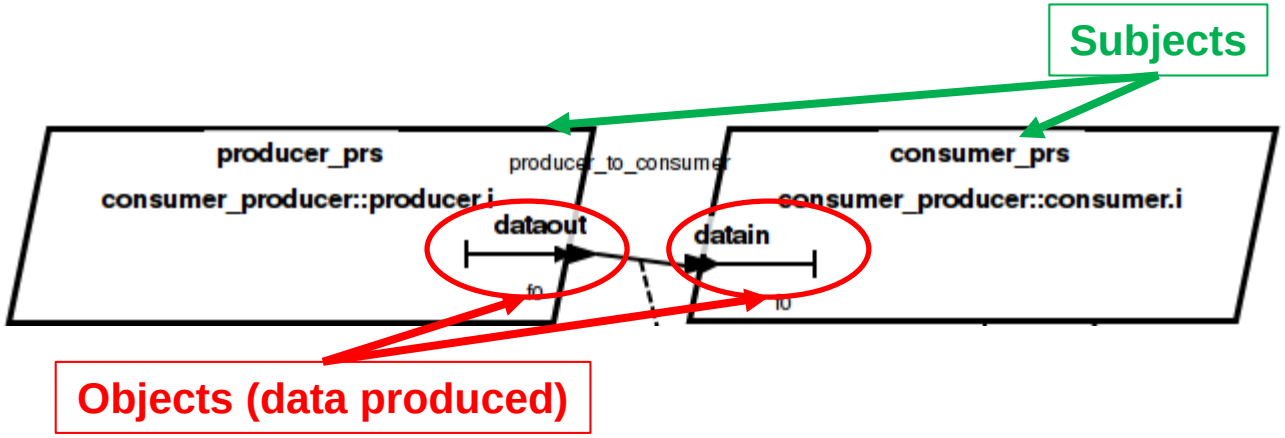
Security Specifications

Leverage AADL properties for security level specification

- Define security-specific values
- Associate them with components and interfaces

Direct mapping to MILS Security Level concepts

- MILS subjects to AADL runtime components
- MILS objects to AADL interfaces



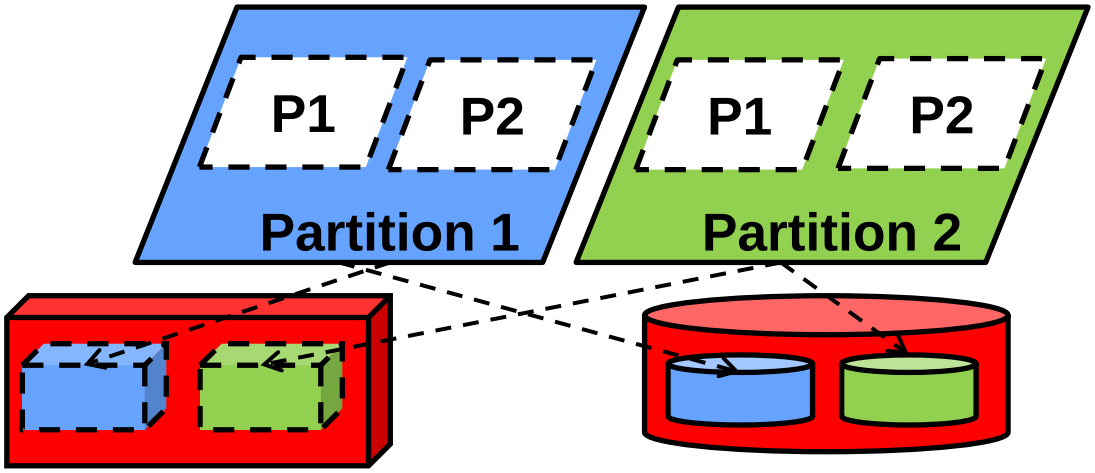
Partitioning Policy (as in ARINC653 or MILS)

Partitions content and attributes

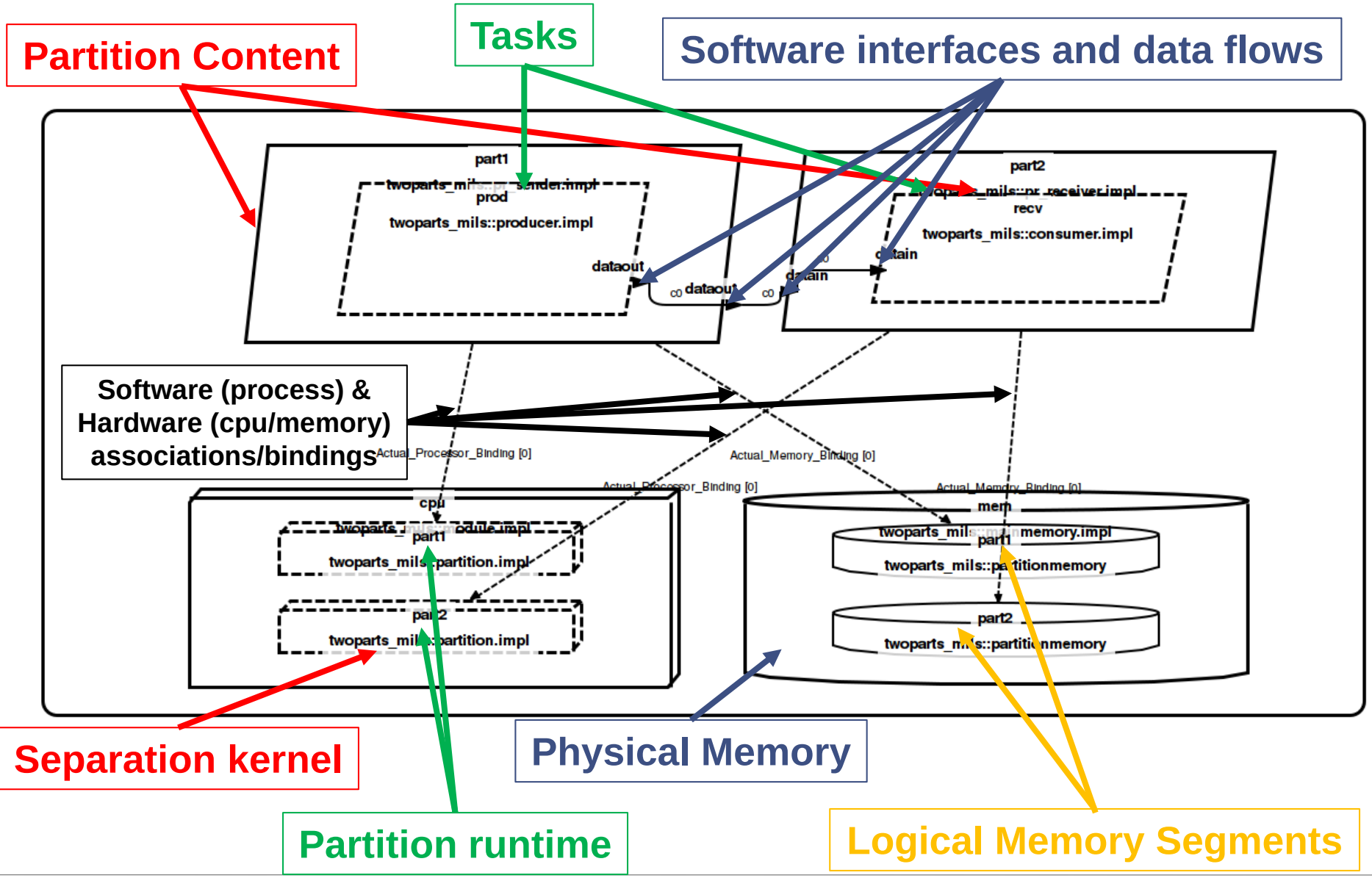
- Use the regular process component
- Include partition resources (tasks, data, etc.)

Time and Space Isolation

- Time: Partition execution slots
- Space: Association of partitions to memory segments



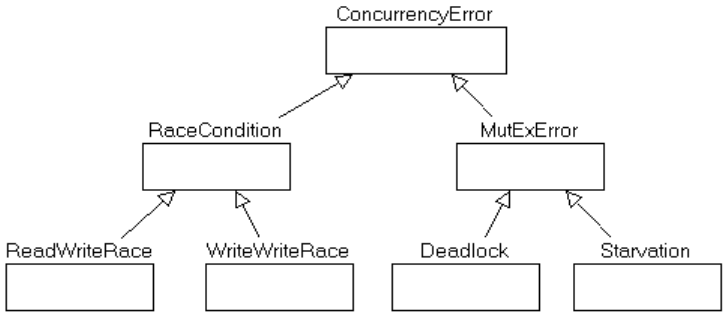
Modeling a MILS architecture - example



Safety Policy with the Error-Model Annex V2

Standardized AADL annex dedicated for safety specification

- Integrated with AADL-core
- Extend/refine existing models



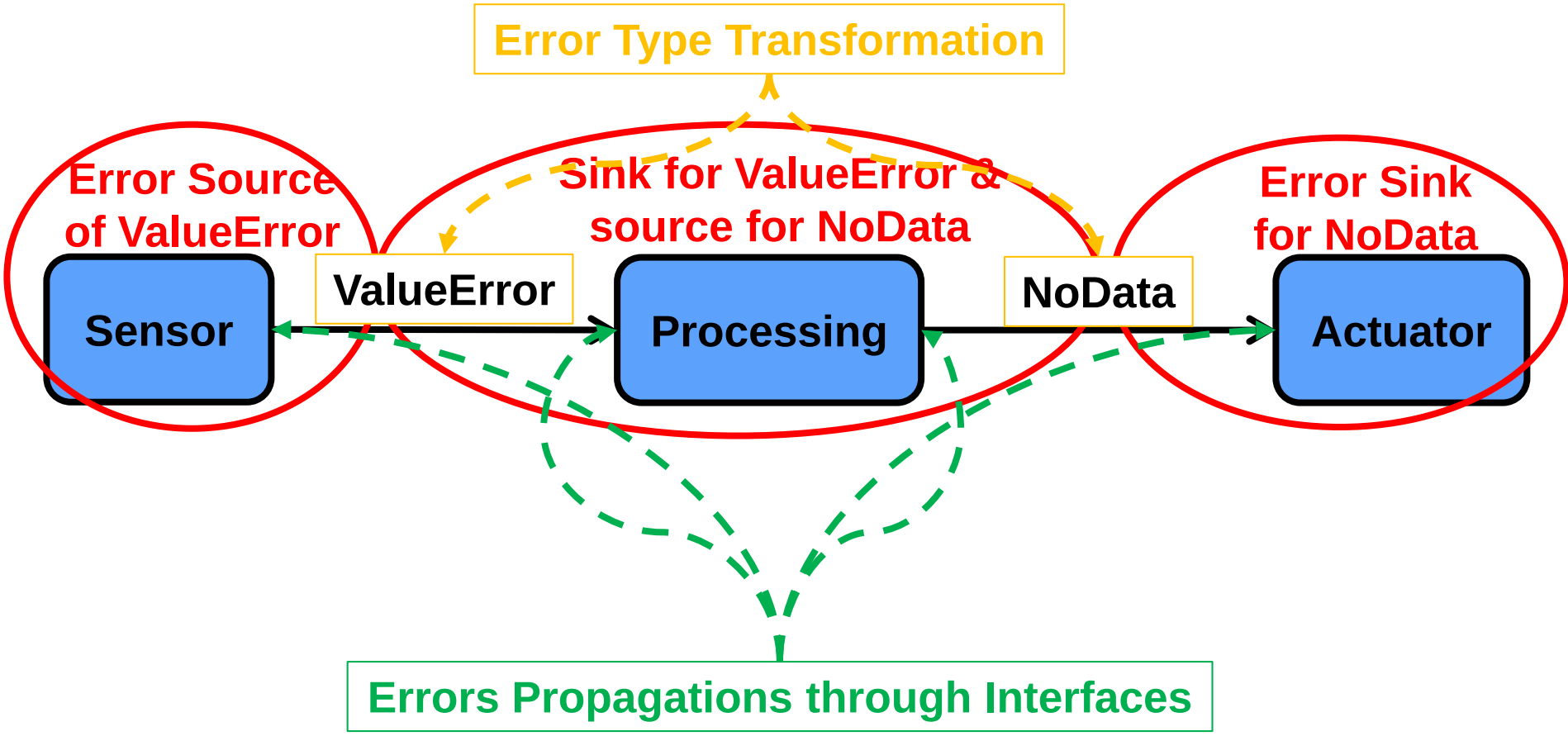
Support of Error Types Ontology

- Characterize the error (i.e. divide by zero, late value)
- Types hierarchy (i.e. late value is an extension of a timing error)

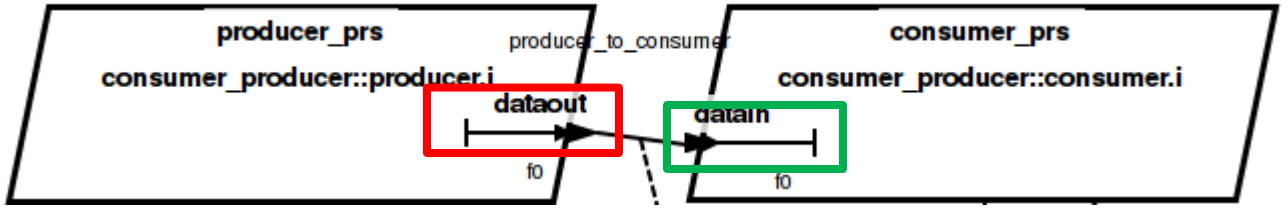
Error Propagations and Behavior Specification

- Errors being propagated by AADL components
- Behavior based on external interfaces or sub-components

Error Propagation



Error Propagation Example




```
thread producer
features
    dataout : out data port Character;
annex EMV2 {**
    use types errorlibrary;
    use behavior errorlibrary::FailAndRecover;
    error propagations
        dataout : out propagation {ValueError};
    flows
        f0 : error source dataout {ValueError};
    end propagations;

    component error behavior
    events
        ComputationError : error event;
    transitions
        t0 : Operational -[ComputationError]-> Failed;
    propagations
        p0 : Failed -[]-> dataout{ValueError};
    end component;

    properties
        EMV2::severity => ARP4761::Hazardous applies to dataout.ValueError;
        EMV2::OccurrenceDistribution => [ ProbabilityValue => 1.42e-5 ; Distribution => Poisson;]
            applies to dataout.ValueError;
        EMV2::likelihood => ARP4761::Probable applies to dataout.ValueError;
        EMV2::hazards =>
            ([  crossreference => "TBD";
                failure => "";
                phases => ("all");
                description => "Bad Value from the thread producer";
                comment => "Must check the software that the value is not faulty";
            ])
            applies to dataout.ValueError;
**};
```

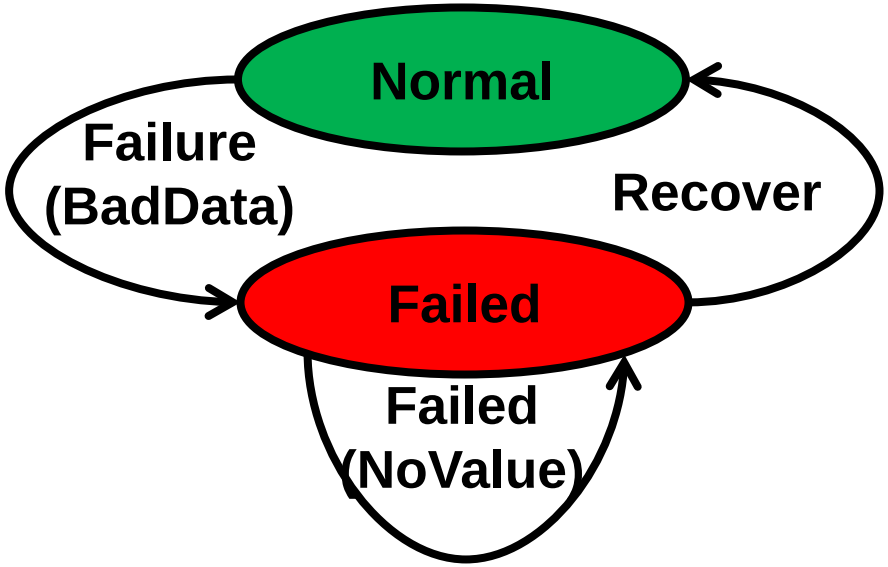
```
thread consumer
features
    datain : in data port Character;
annex EMV2 {**
    use types errorlibrary;
    use behavior errorlibrary::FailAndRecover;
    error propagations
        datain : in propagation {ValueError};
    flows
        f0 : error sink datain {ValueError};
    end propagations;

    component error behavior
    transitions
        t0 : Operational -[datain{ValueError}]-> Failed;
    end component;
    properties
        EMV2::severity => ARP4761::Hazardous applies to datain.ValueError;
**};
end consumer;
```

Error behavior

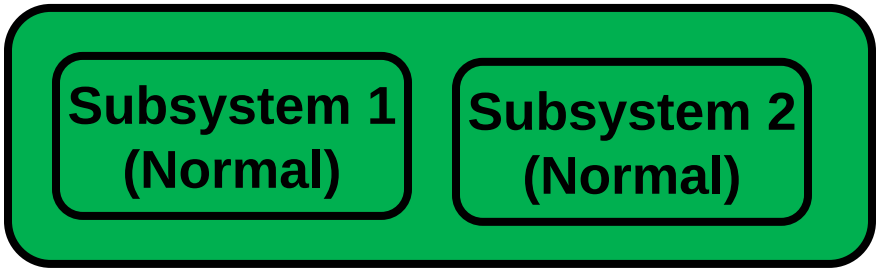
States machines

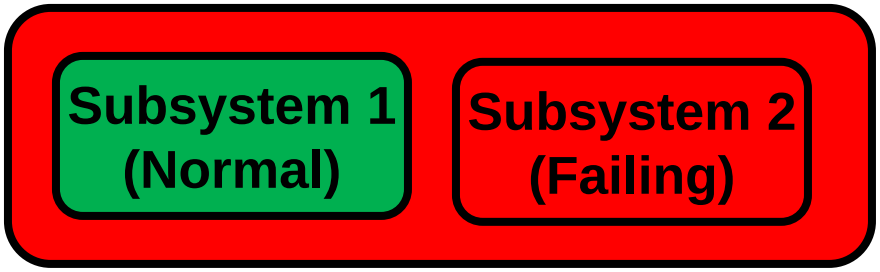
- Error-related transitions
- Propagation rules
- Use of error types



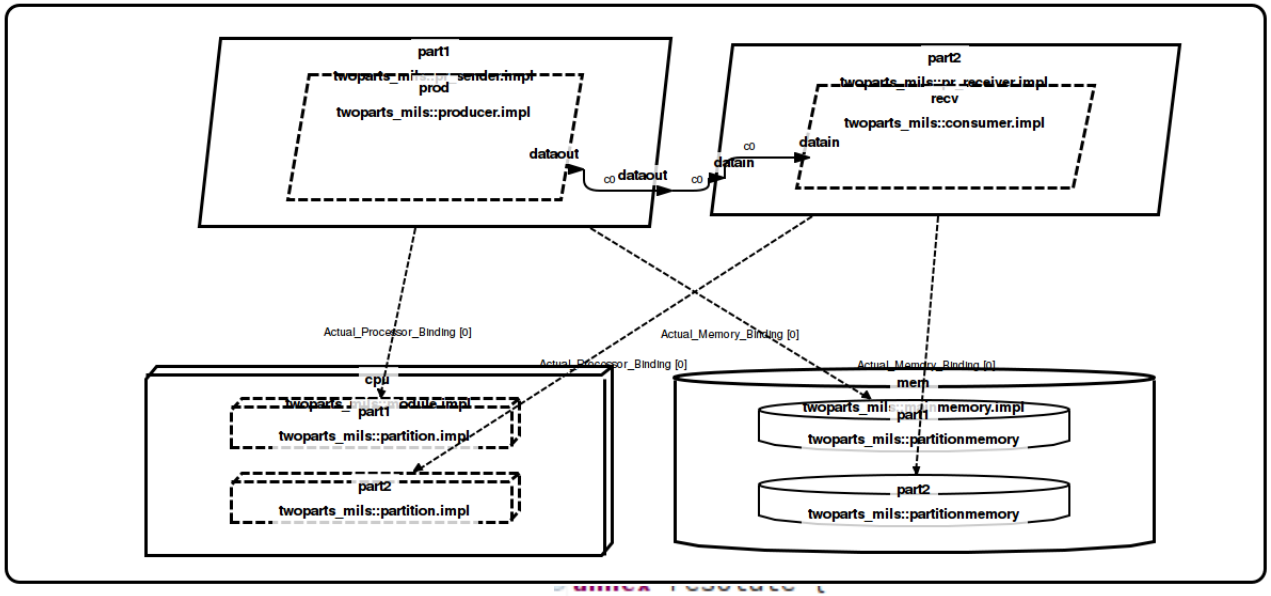
Composite behavior

- Define system states according to its parts
- *ex: “I am failing if one of my component is failing”*





Error behavior example




```
:pu.part1)) applies to part1;
:pu.part2)) applies to part2;

't1;

't2;
```

```
annex resolute {
--   prove (check_deos_compliance(this))
     prove (check_mils_compliance(this))
**};
annex EMV2 {**
     use types errorlibrary;
     use behavior errorlibrary::FailAndRecover;

     composite error behavior
     states
          [part1.Failed]-> Failed;
          [part2.Failed]-> Failed;
          [cpu.Failed]-> Failed;
     end composite;
**};
end node.impl;
```

Agenda

Introduction to AADL

AADL modeling patterns for safety and security

AADL validation tools dedicated to security and safety

Demonstration

Security Policy Verification

Component integration and composition

- Partitions share the same level with their tasks
- Partitions contain objects at the same level

Runtime issues

- Each process is isolated in a partition
- Partitions has at least one execution slot
- Memory segments contain partitions at the same security level

Communication Policies

- Communication share the same level
- A shared device manages objects at the same level

Specifying Validation Rules with RESOLUTE

Specify constraints on the AADL model

Check model consistency and properties

Validation at model level, avoid propagation of errors

List of rules and functions to check the model

Select elements to be verified

Filter them according to your constraints

Check components characteristics

```
check_mils_partitions_connections (s : system) <=
    ** "Check that connected partitions in " s " share the same security level" **
    forall (p1 : process) (p2 : process) (c : connection) (vp1 : virtual_processor) (vp2 : virtual_processor) .
    (connected (p1, c, p2)) and (processor_bound (p1, vp1)) and (processor_bound (p2, vp2))
    => property (vp1, SEI::SecurityLevel) = property (vp2, SEI::SecurityLevel)
```

Select process, connections & virtual processor elements

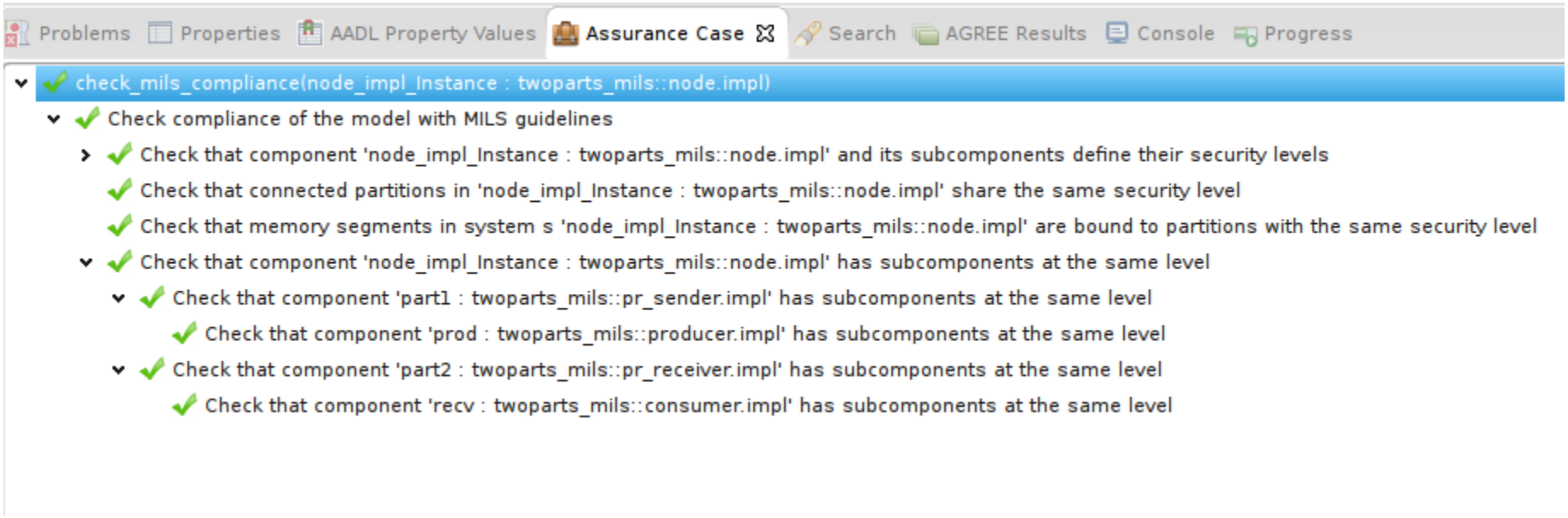
Filter connected partitions with their associated runtime

Check the runtime security level is equal

Generating Assurance Cases

Generate assurance-cases using RESOLUTE and AADL

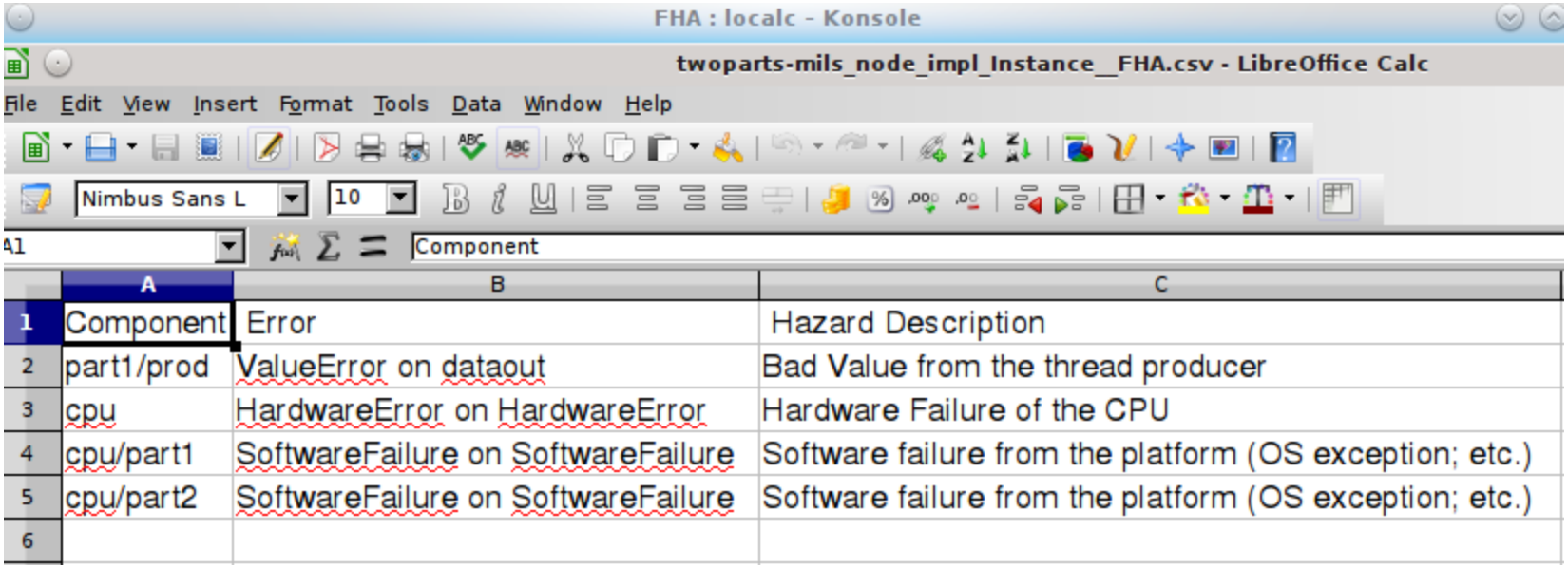
- Show constraints dependencies
- Export to Certware



Safety documentation Generation - FHA

Functional Hazard Assessment

List of all error sources of the system



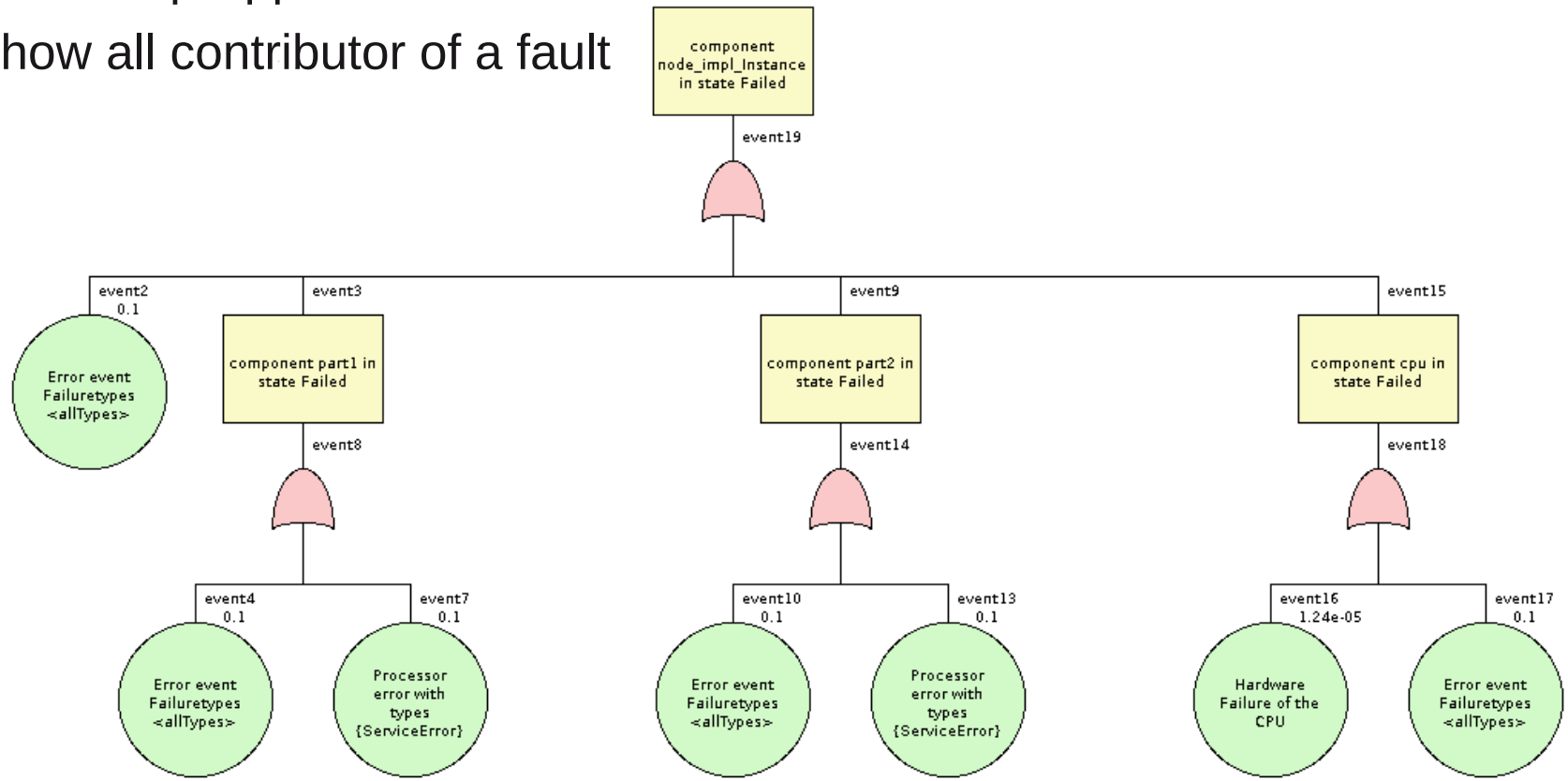
	A	B	C
1	Component	Error	Hazard Description
2	part1/prod	ValueError on dataout	Bad Value from the thread producer
3	cpu	HardwareError on HardwareError	Hardware Failure of the CPU
4	cpu/part1	SoftwareFailure on SoftwareFailure	Software failure from the platform (OS exception; etc.)
5	cpu/part2	SoftwareFailure on SoftwareFailure	Software failure from the platform (OS exception; etc.)
6			

Safety documentation Generation - FTA

Fault-Tree Analysis

Bottom-up Approach

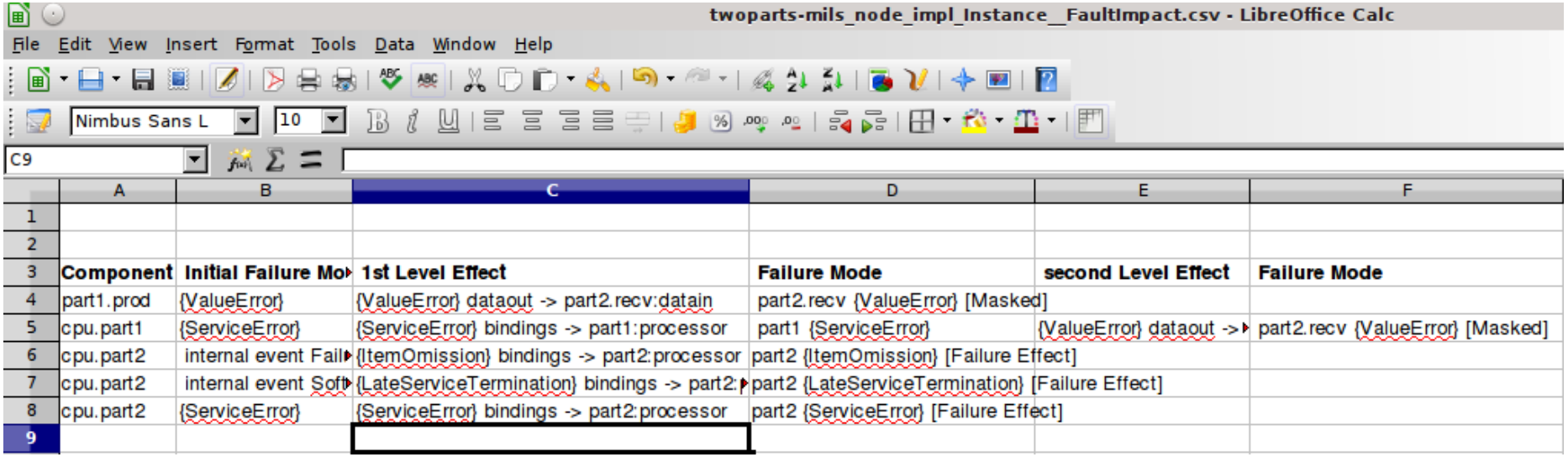
Show all contributor of a fault



Safety documentation Generation – Fault Impact

Failure Mode and Effect Analysis

- Propagation paths of failures
- Highlight failure containment



	A	B	C	D	E	F
1						
2						
3	**Component**	**Initial Failure Mo**	**1st Level Effect**	**Failure Mode**	**second Level Effect**	**Failure Mode**
4	part1.prod	{ValueError}	{ValueError} dataout -> part2.recv:datain	part2.recv {ValueError} [Masked]		
5	cpu.part1	{ServiceError}	{ServiceError} bindings -> part1:processor	part1 {ServiceError}	{ValueError} dataout ->	part2.recv {ValueError} [Masked]
6	cpu.part2	internal event Fail	{ItemOmission} bindings -> part2:processor	part2 {ItemOmission} [Failure Effect]		
7	cpu.part2	internal event Soft	{LateServiceTermination} bindings -> part2:	part2 {LateServiceTermination} [Failure Effect]		
8	cpu.part2	{ServiceError}	{ServiceError} bindings -> part2:processor	part2 {ServiceError} [Failure Effect]		
9						

Automatic Code Generation

Automatically produce system implementation

- Ensure implementation of system requirements
- Avoid traditional mistakes of manual code generation

Low overhead (memory footprint and additional CPU time)

- Less than 10% in memory and computation increase
- Benefits outweigh the potential

Support for different runtime

- ARINC653/MILS – focus on safety/security (DeOS, POK)
- POSIX (RTEMS, Linux)

Agenda

Introduction to AADL

AADL modeling patterns for safety and security

AADL validation tools dedicated to security and safety

Demonstration

Conclusion

AADL flexible language to define safety and security concerns

- Early verification, reducing tests and integration costs
- Automatic code production, avoiding code and integration mistakes

Integration with existing development methods

- Safety documentation (i.e. ARP4761)
- Coding standards (i.e. ARINC653)

Bridge with Validation and Assurance Case tools

- Check model consistency and composition
- Auto-Generate assurance cases from models

Links & Useful Information

AADL website – http://www.aadl.info

AADL wiki – http://www.aadl.info/wiki

ARINC653 AADL annex standard - http://standards.sae.org/as5506/2/

Contact

Dr. Julien Delange
RTSS AP Initiative
Telephone: +1 412-268-9652
Email: jdelange@sei.cmu.edu

U.S. Mail
Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA

Web
www.aadl.info
www.sei.cmu.edu
www.sei.cmu.edu/contact.cfm

Customer Relations
Email: info@sei.cmu.edu
Telephone: +1 412-268-5800
SEI Phone: +1 412-268-5800
SEI Fax: +1 412-268-6257